Computer Architecture

Exercise 5

Due 5. December 2008, 5:00pm

The following sources shall increase your understanding about Tomasulo's Algorithm..

- R. M. Tomasulo - *An Efficient Algorithm For Exploiting Multiple Arithmetic Units*
  `http://www.research.ibm.com/journal/rd/111/tomasulo.pdf`
- UMASS - *Simulation of the Tomasulo's algorithm used for Dynamic Scheduling*
  `http://www.ecs.umass.edu/ece/koren/architecture/Tomasulo1/tomasulo.htm`

Part 1. *Case Study - Register Renaming, Dynamic Scheduling, Multiple Issue Designs*

1.1. Assume a five-stage single-pipeline microarchitecture (fetch, decode, execute, memory, write back) and the code below. All ops are 1 cycle except LW and SW, which are 1 + 2 cycles, and branches, which are 1 + 1 cycles. There is no forwarding. Given is the following example code:

```
Loop    LW      R1, 0(R2)
I0      ADDI    R1, R1, #1
I1      SW      R1, 0(R2)
I2      ADDI    R2, R2, #4
I3      SUB     R4, R3, R2
I4      BNZ     R4, Loop
```

Note: An instruction does not enter the execution phase until all of its operands are ready. The decode and write-back can overlap. Branch overhead is the amount of cycles needed to decide, if a branch is taken or not (begining with the fetch).

a) Show the phases of each instruction per clock cycle for one iteration of the loop.

b) How many clock cycles per loop iteration are lost to branch overhead?

c) Assume a static branch predictor, capable of recognizing a backwards branch in the Decode stage (heuristic: $PC_{branch\_target} < PC_{actual}$ –> loop, take branch). Now how many clock cycles are wasted on branch overhead?

d) Assume a dynamic branch predictor. How many cycles are lost on a correct prediction?

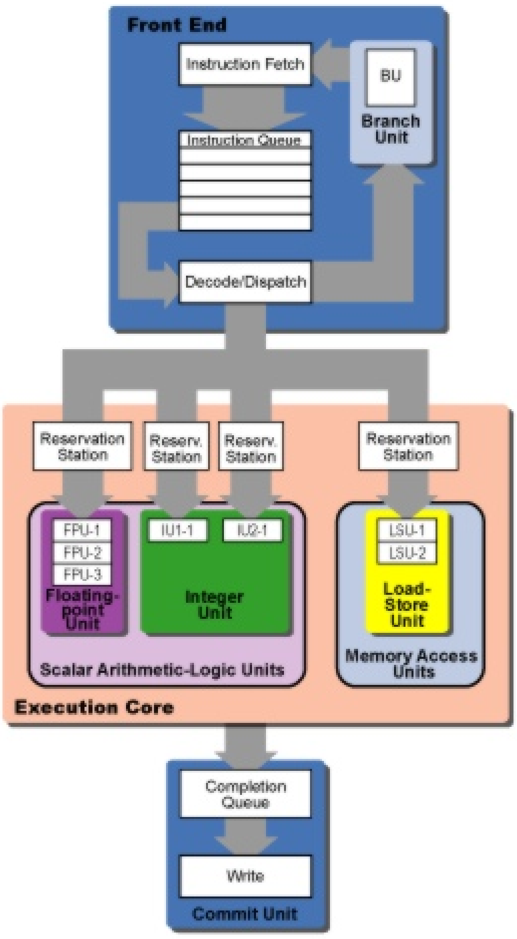

Figure 1: Power PC 750 (fetchs up to four instructions per cycle and decodes up to two non-bracnch instruction per cycle from the instruction queue)

1.2. Let's consider what dynamic scheduling might achieve here. Assume a microarchitecture as shown in figure 2. Assume that the ALUs can do all arithmetic ops (MULTD, DIVD, ADDD, ADDI, SUB) and branches, and that the Reservation Station (RS) can dispatch at most one operation to each functional unit per cycle (one op to each ALU plus one memory op to the LD/ST unit).

a) Suppose all of the instructions from the sequence are present in the RS, with no renaming having been done. Highlight any instructions in the code where register renaming would improve performance (that is, instructions can be dispatched earlier or resolve multiple copies of an operand). Hint: Look for RAW and WAW hazards. Assume the functional unit latencies as in table 1. Given is the following example code:

```
Loop:   LD     F2, 0(Rx)
I0.     MULTD  F2, F0, F2
I1.     DIVD   F8, F2, F0
I2.     LD     F4, 0(Ry)
I3.     ADDD   F4, F0, F4
I4.     ADDD   F10, F8, F2
I5.     SD     F4, 0(Ry)
I6.     ADDI   Rx, Rx, #8
I7.     ADDI   Ry, Ry, #8
I8.     SUB    R20, R4, Rx
I9.     BNZ    R20, Loop
```

Table 1: Latencies.

Instruction	**Latency beyond single cycle**
Memory LD	+3
Memory SD	+1
Integer ADD, SUB	+0
Branches	+1
ADDD	+2
MULTD	+4
DIVD	+10

b) Suppose the register-renamed version of the code from a) is resident in the RS in clock cycle N, with latencies as given in table 1. Show how the RS should dispatch these instructions out-of-order, clock by clock, to obtain optimal performance on this code. (Assume the same RS restrictions as in a). Also assume that results must be written into the RS before they're available for use; i.e., no bypassing.) How many clock cycles does the code sequence take?

c) Part b) lets the RS try to optimally schedule these instructions. But in reality, the whole instruction sequence of interest is not usually present in the RS. Instead,

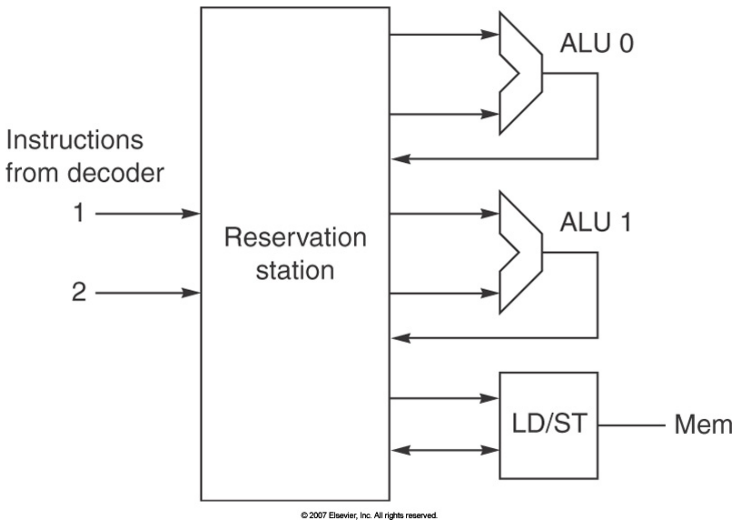
Instructions from decoder
1
2
Reservation station
ALU 0
ALU 1
LD/ST
Mem
© 2007 Elsevier, Inc. All rights reserved.

Figure 2: An out-of-order microarchitecture

various events clear the RS, and as a new code sequence streams in from the decoder, the RS must choose to dispatch what it has. Suppose that the RS is empty. In cycle 0 the first two register-renamed instructions of this sequence appear in the RS. Assume it takes 1 clock cycle to dispatch any op, and assume functional unit latencies are as in table 1. Further assume that the front end (decoder/register-renamer) will continue to supply two new instructions per clock cycle. Show the cycle-by-cycle order of dispatch of the RS. How many clock cycles does this code sequence require now?

d) If you wanted to improve the results of part c), which would have helped most: (1) another ALU; (2) another LD/ST unit; (3) cutting the longest latency in half? What's the speedup?

e) Now let's consider speculation, the act of fetching, decoding, and executing beyond one or more conditional branches. Our motivation to do this is twofold: the dispatch schedule we came up with in part c) had lots of nops, and we know computers spend most of their time executing loops (which implies the branch back to the top of the loop is pretty predictable.) Loops tell us where to find more work to do; our sparse dispatch schedule suggests we have opportunities to do some of that work earlier than before. In part d) you found the critical path through the loop. Imagine folding a second copy of that path onto the schedule you got in part b). How many more clock cycles would be required to do two loops' worth of work (assuming all instructions are resident in the RS)? (Assume all functional units are fully pipelined.)

Part 2. *Home Work - Register Renaming, Dynamic Scheduling, Multiple Issue Designs [10 pt.]*

Let's consider what dynamic scheduling might achieve here. Assume a microarchitecture as shown in figure 2. Assume that the ALUs can do all arithmetic ops (MULTD, DIVD, ADDD, ADDI, SUB) and branches, and that the Reservation Station (RS) can dispatch at most one operation to each functional unit per cycle (one op to each ALU plus one memory op to the LD/ST unit).

a) Suppose all of the instructions from the sequence are present in the RS, with no renaming having been done. Highlight any instructions in the code where register renaming would improve performance (that is, instructions can be dispatched earlier or resolve multiple copies of an operand). Hint: Look for RAW and WAW hazards. Assume the functional unit latencies as in table 2. [1 pt.]
Given is the following example code:

```
Loop:   LD     F2,  0(Rx)
I0.     DIVD   F8,  F2,  F0
I1.     MULTD  F2,  F6,  F2
I2.     LD     F4,  0(Ry)
I3.     ADDD   F4,  F0,  F4
I4.     ADDD   F10, F8,  F2
```

```
I5.    ADDI   Rx, Rx, #8
I6.    ADDI   Ry, Ry, #8
I7.    SD     F4, 0(Ry)
I8.    SUB    R20, R4, Rx
I9.    BNZ    R20, Loop
```

Table 2: Latencies.

Instruction	Latency beyond single cycle
Memory LD	+4
Memory SD	+1
Integer ADD, SUB	+0
Branches	+1
ADDD	+1
MULTD	+5
DIVD	+12

b) Suppose the register-renamed version of the code from a) is resident in the RS in clock cycle N, with latencies as given in table 2. Show how the RS should dispatch these instructions out-of-order, clock by clock, to obtain optimal performance on this code. (Assume the same RS restrictions as in a). Also assume that results must be written into the RS before they're available for use; i.e., no bypassing.) How many clock cycles does the code sequence take? [2 pt.]

c) Part b) lets the RS try to optimally schedule these instructions. But in reality, the whole instruction sequence of interest is not usually present in the RS. Instead, various events clear the RS, and as a new code sequence streams in from the decoder, the RS must choose to dispatch what it has. Suppose that the RS is empty. In cycle 0 the first two register-renamed instructions of this sequence appear in the RS. Assume it takes 1 clock cycle to dispatch any op, and assume functional unit latencies are as in table 2. Further assume that the front end (decoder/register-renamer) will continue to supply two new instructions per clock cycle. Show the cycle-by-cycle order of dispatch of the RS. How many clock cycles does this code sequence require now? [3 pt.]

d) If you wanted to improve the results of part c), which would have helped most: (1) another ALU; (2) another LD/ST unit; (3) full bypassing of ALU results to subsequent operations; (4) cutting the longest latency in half? What's the speedup? [2 pt.]

e) Now let's consider speculation, the act of fetching, decoding, and executing beyond one or more conditional branches. Our motivation to do this is twofold: the dispatch schedule we came up with in part c) had lots of nops, and we know computers spend most of their time executing loops (which implies the branch back to the top of the loop is pretty predictable.) Loops tell us where to find more work to do; our sparse dispatch schedule suggests we have opportunities to do some of that work earlier than before. In part d) you found the critical path through the loop. Imagine folding a second copy of that path onto the schedule you got in part b). How many more clock cycles would be required to do two loops' worth of work (assuming all instructions are resident in the RS)? (Assume all functional units are fully pipelined.) [2 pt.]

Part 3. *Home Work - Scoreboarding and Tomasulo's Approach [5 pt.]*

Both Scoreboarding and Tomasulo's Approach allow out-of-order execution. Go through the slides again and answer the following questions.

a) How does the Scoreboard Control solve WAW and WAR hazards? [1 pt.]

b) How does Tomasulo's Approach solve WAW and WAR hazards? [1 pt.]

c) What are the main differences between Scoreboarding and Tomasulo's Approach? [1 pt.]

d) What is the Common Data Bus for and what is his benefit? [1 pt.]

e) What problem occurs, if two functional units finish their computation and write their results? [1 pt.]

Your solutions needs to be be traceable!